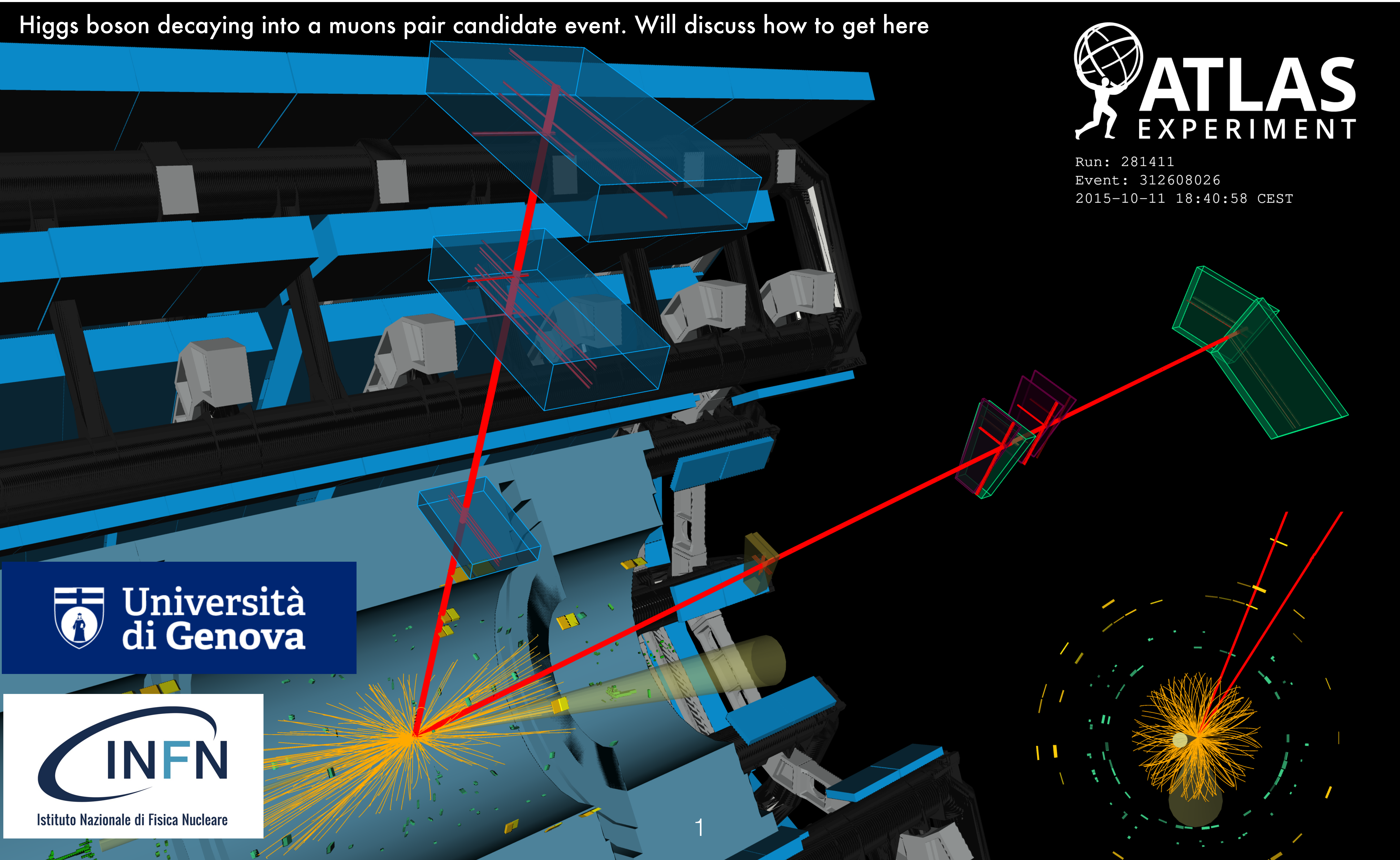
Studies on track finding algorithms based on machine learning with GPU and FPGA

F.A. Di Bello on behalf of the ATLAS TDAQ collaboration

Real Time IEEE 2024

Higgs boson decaying into a muons pair candidate event. Will discuss how to get here



Università di Genova

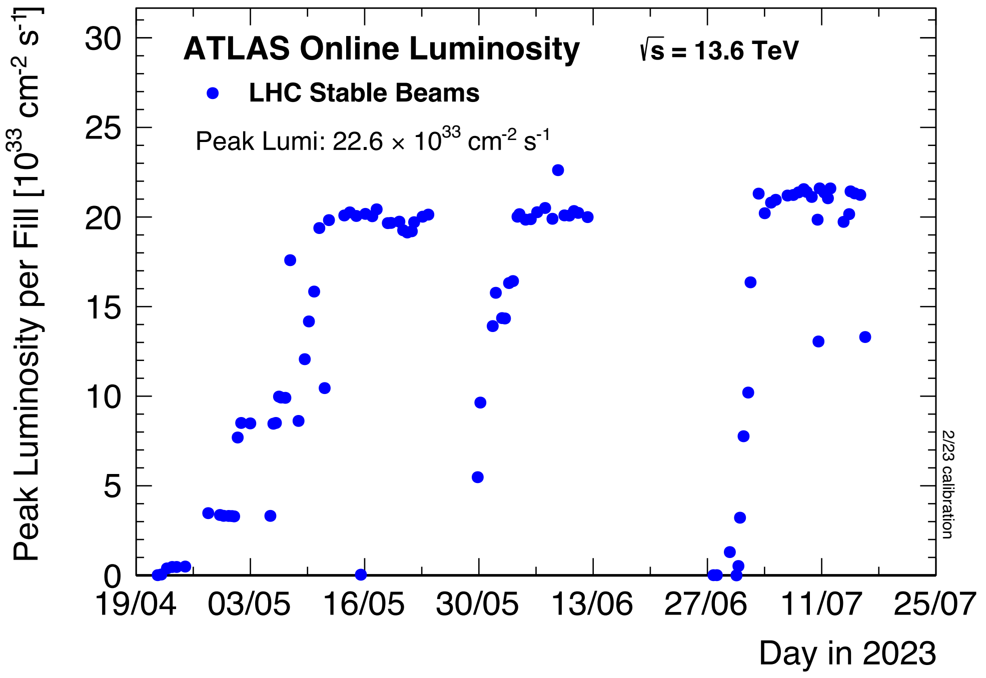
INFN
Istituto Nazionale di Fisica Nucleare

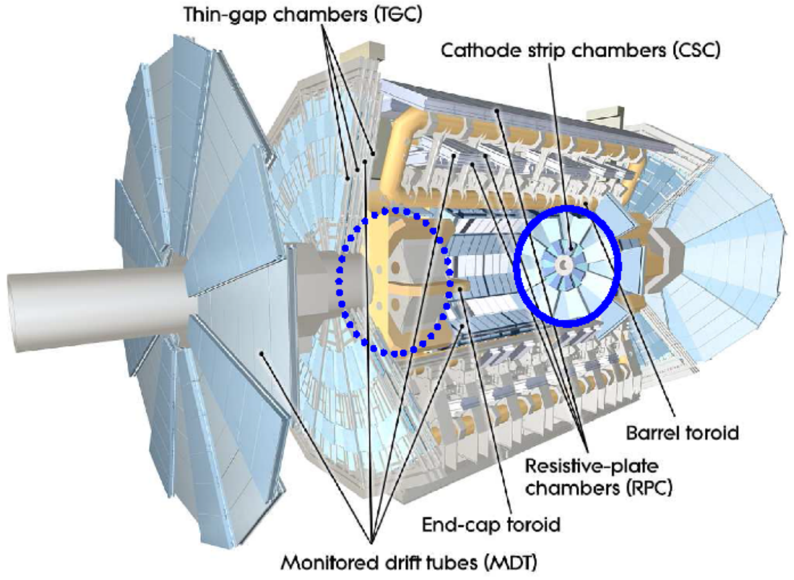
Introduction

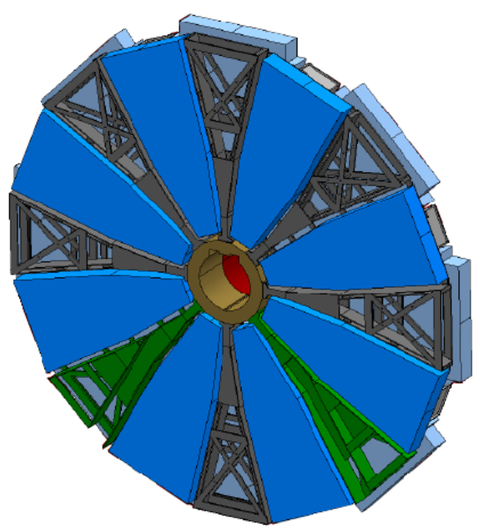
- The ATLAS experiment is presently successfully collecting data
- Strong effort to upgrade its trigger system from RUN2 to RUN3
- Special effort particularly towards muon trigger upgrades, where a new detector, the small wheel has been installed

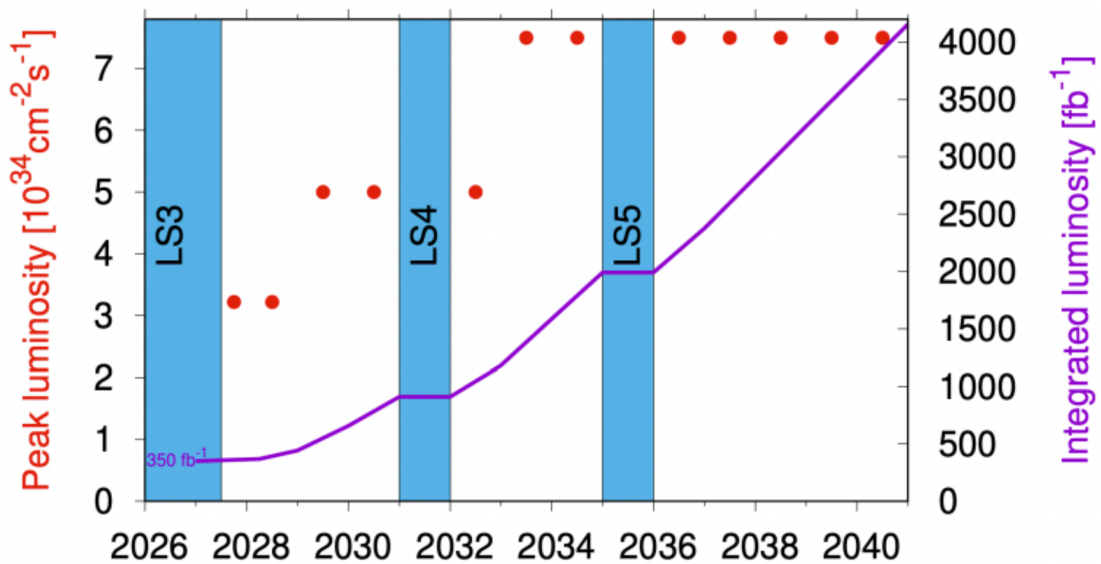
The main object of the talk:

- Discuss possibility to include ML algorithms for muon tracking
- At the HL-LHC, an heterogeneous high-level triggering farm is considered, compromise between performance, costs, power-consumption



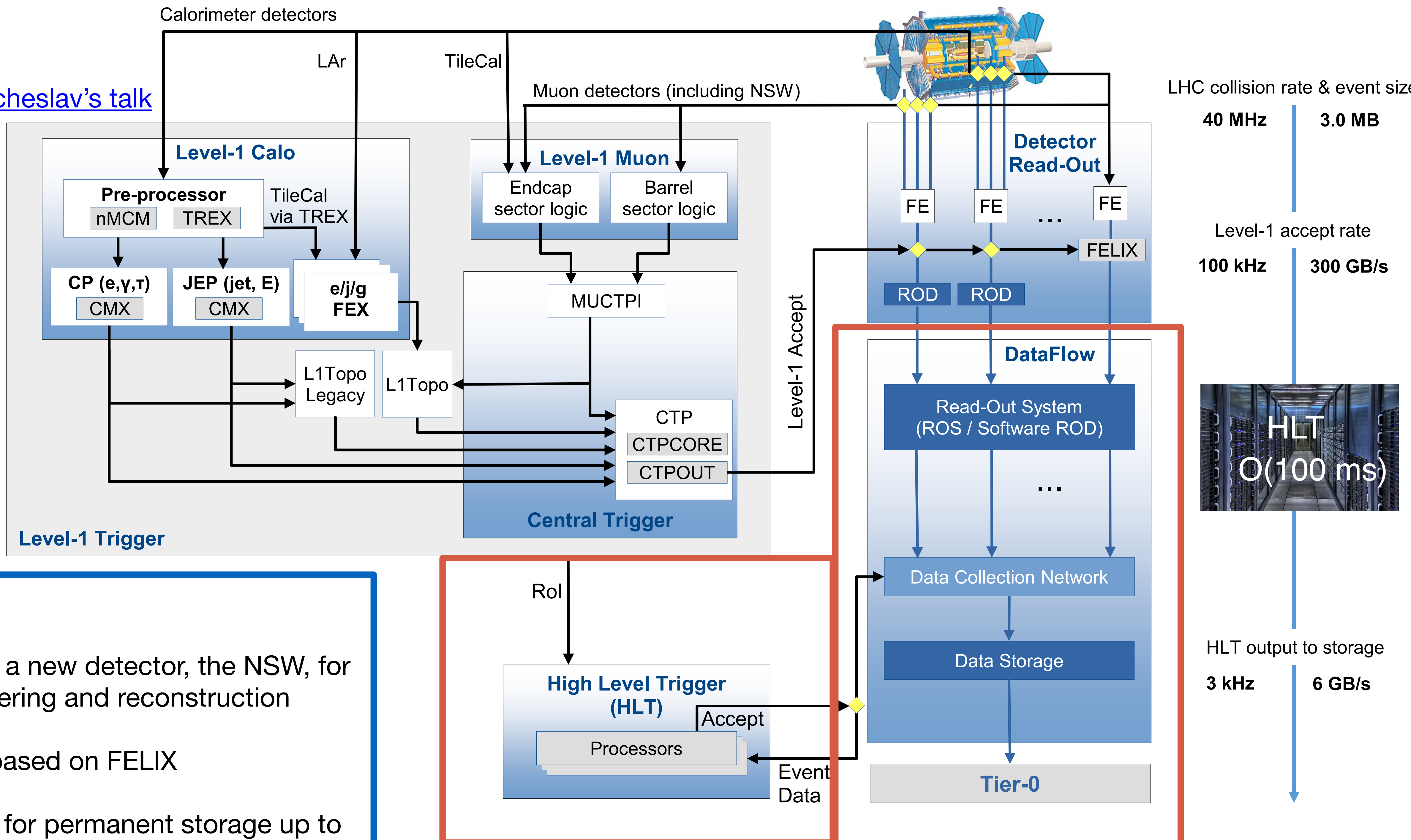






The ATLAS trigger system in RUN3

[Viachesla's talk]



- Addition of a new detector, the NSW, for muon triggering and reconstruction
- Read-out based on FELIX
- Bandwidth for permanent storage up to 8 Gb/s (higher than nominal at 6 Gb/s)
- ATLAS Event Filter farm will migrate to a heterogeneous system of CPU/GPU/ FPGA for the HL-LHC [TDR]

Aim of the talk:

Can accelerator cards commercially designed for machine learning application be helpful for the muon trigger system?

The toy model used in this study

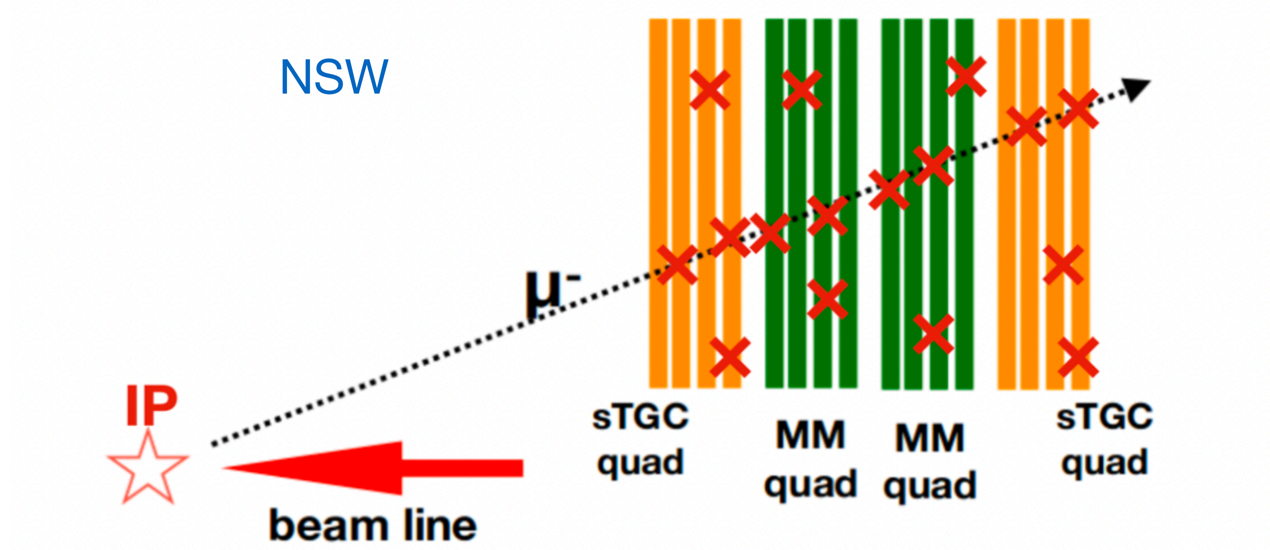
To speed up R&D part of the study, a toy model is simulated

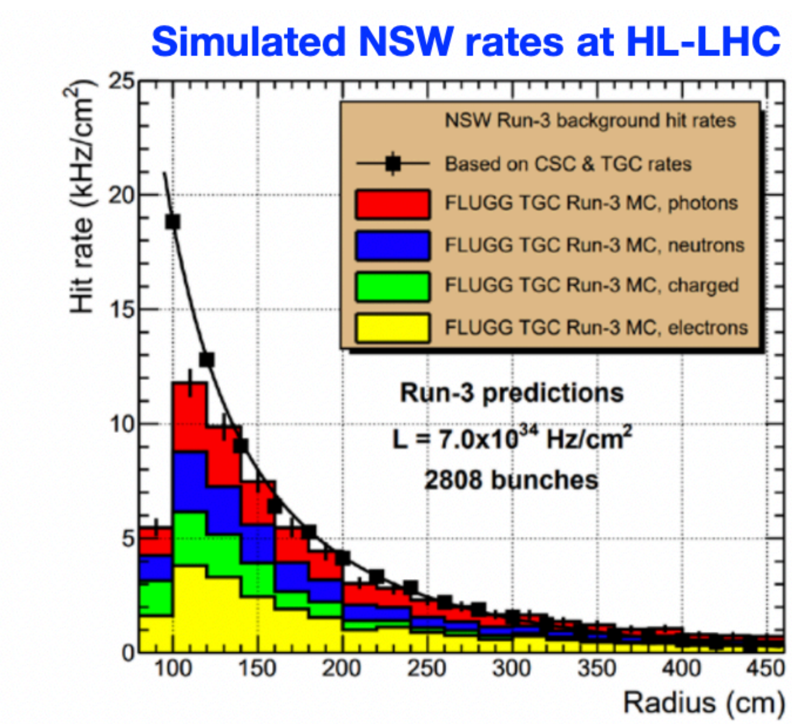
Geant4 based toy model for a generic detector, inspired by the NSW geometry

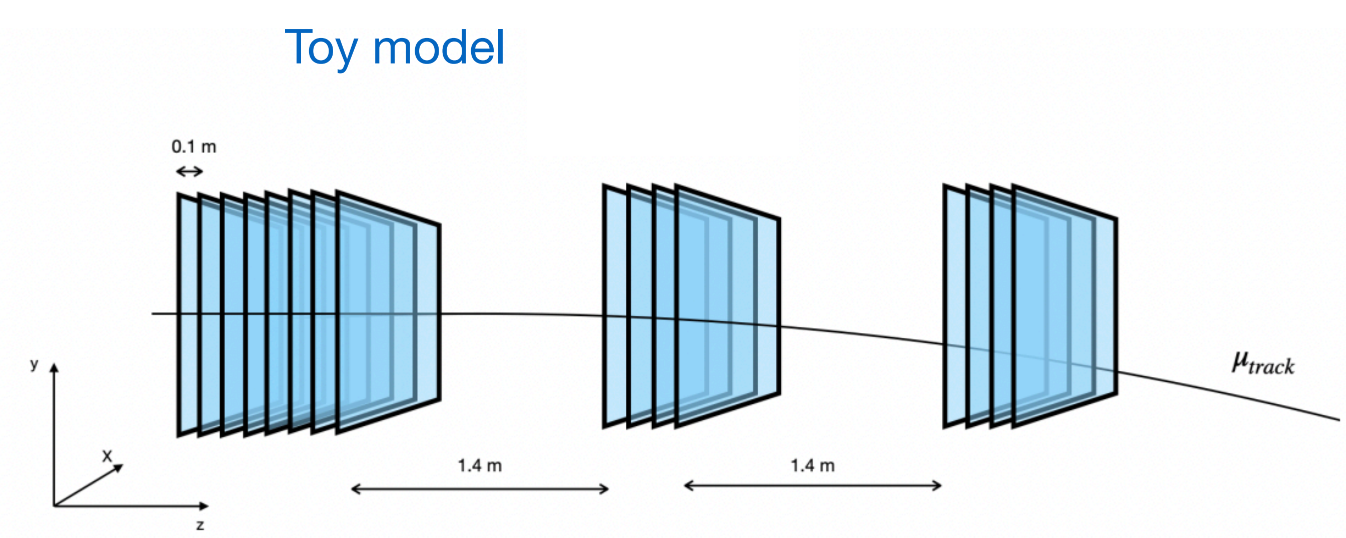
Different noise rate are tested: 2, 5, 10, 15 kHz/cm^2 (affects the occupancy of single events)

Samples produced with O(10M) events

Effects from correlated background is also emulated: e.g. electrons from material interactions downstream the detector

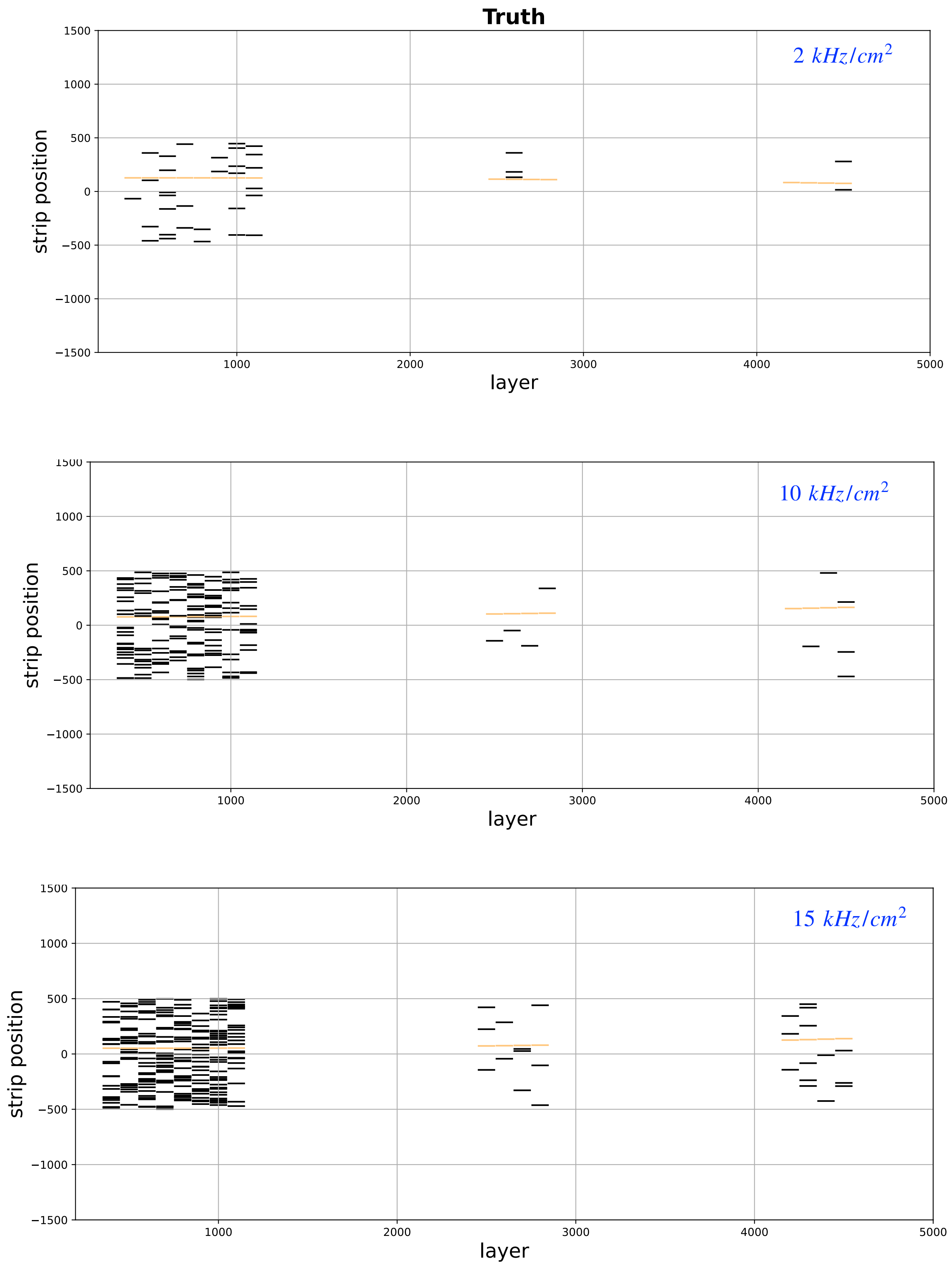






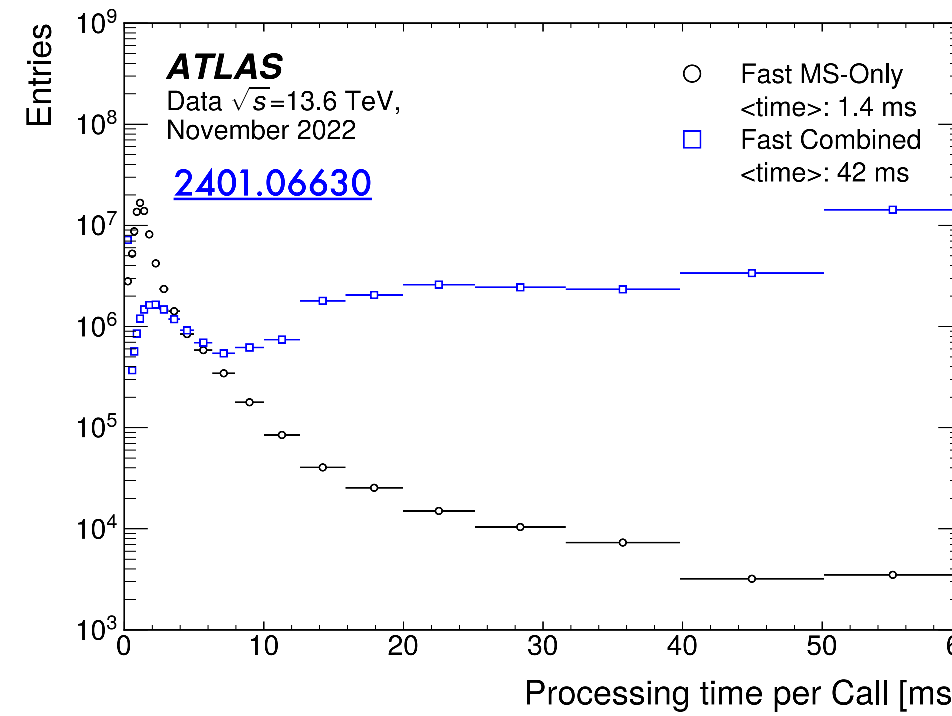
Occupancy examples

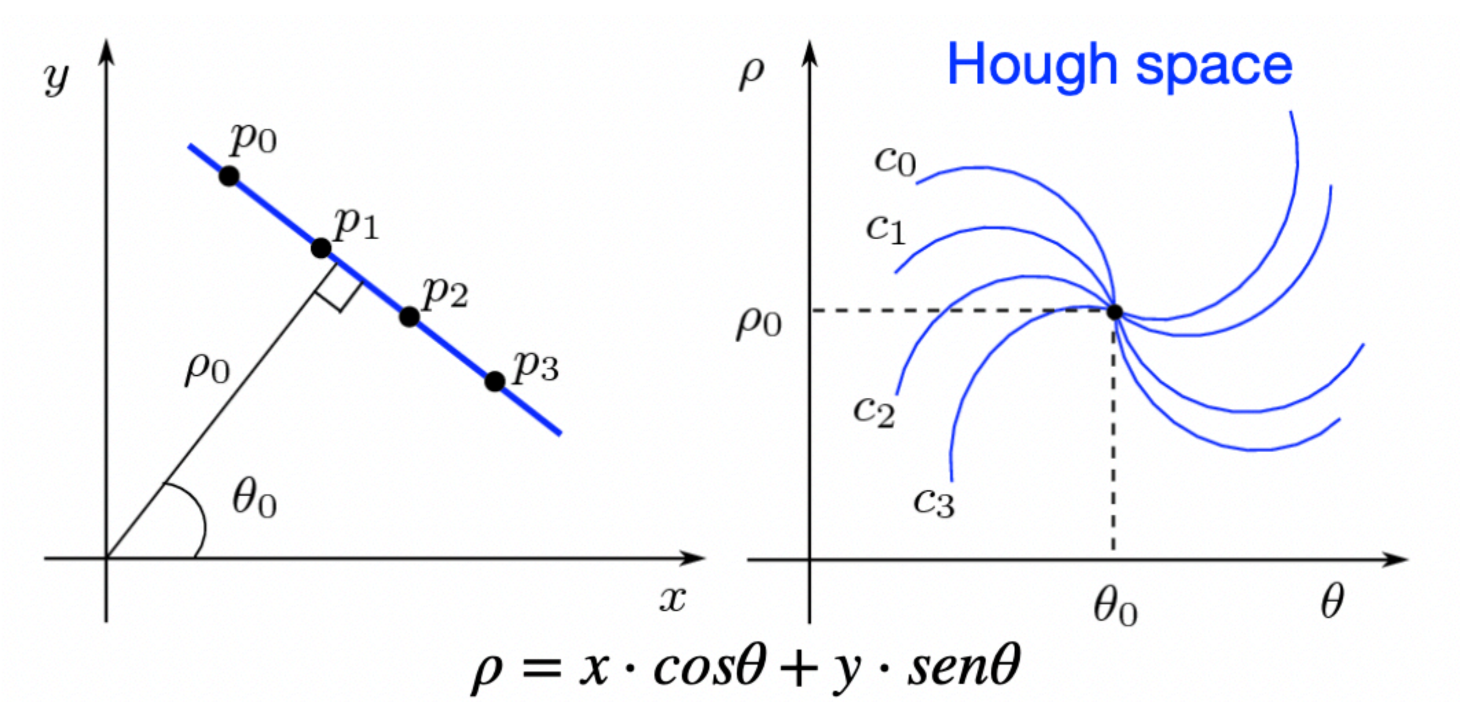
Will not discuss how to get predictions

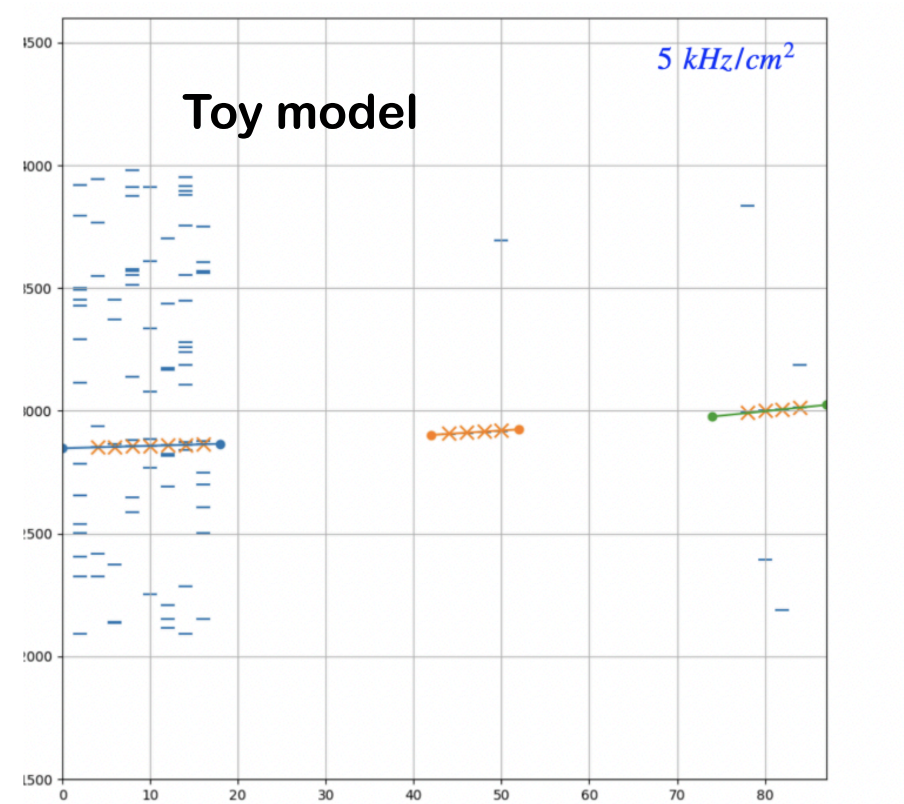


Muon trigger system timing performance

- Trigger algorithms based on Hough transform (HT)
- Standard in muon tracking since several years: simple and performant algorithms, but comes with caveats…
    1. High level of fine tuning needed: (binning, number of hits in maxima)
    2. Number of fakes increases with occupancy
    3. Number of inference time increases with occupancy







FPGAs usage for machine learning applications

The field of development is very active, with various possibilities available. The typical trade-off lies between customizable solutions, performance, and the simplicity of implementation and maintenance

Mostly relevant for level 1 triggering, discussed during [Vladimir's talk]

Direct HLS implementation into FPGAs:

DIrect implementation

Open source platform developed and maintained for HEP community exists: hls4ml

Main advantage is that is fast and suitable for a level-0 trigger.

Approximate a NN with parametric functional form

A ML model is essentially a function that can be approximated with analytic form. 2305.04099 and pySR4GNN

Easier to implement into FPGAs and no need to load weights explicitly

Need to keep high accuracy, not trivial in general

Only relevant for HLT

Use commercial accelerator cards that offer integrated platform for deployment:

Vitis AI

Adaptable & Real-Time AI Inference Acceleration

No need for fine-tuned maintenance.

High level API - no need to know VHDL/Verilog

Target inference time O(10-100 μs), suitable for HLT

Bounded to the supported architecture

Main focus of this talk

The hardware tested

Xilinx AMD developer several accelerator cards to boost ML inference: cards overview

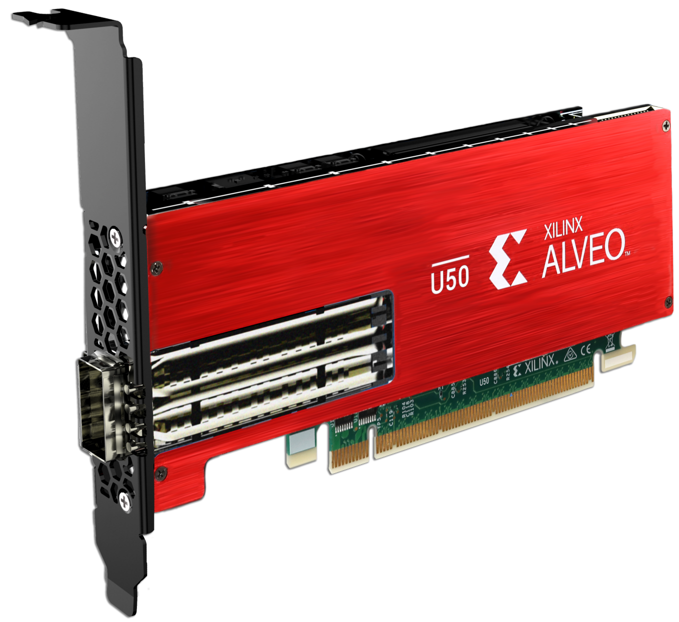
High Level-API: Vitis-AI, more recently also Zebra Mipsology

U250: evaluation card

Based on UltraScale+
LUT: 1728K
Off-Chip DDR memory: 64GB
Off-Chip DDR bandwidth: 77 GB/s
Network Interface: 2x QSFP28
Cost is approximately: 7-10k euro

ML models: DNN and CNN

U50: evaluation card



Based on UltraScale+
LUT: 872k
HBM2 memory: 8GB
HBM2 bandwidth: 316 GB/s
Network Interface: 1 x QSFP28
Cost is approximately: 2-4k euro

ML models: DNN and CNN
* U50LV also supports RNN

VCK5000: development card

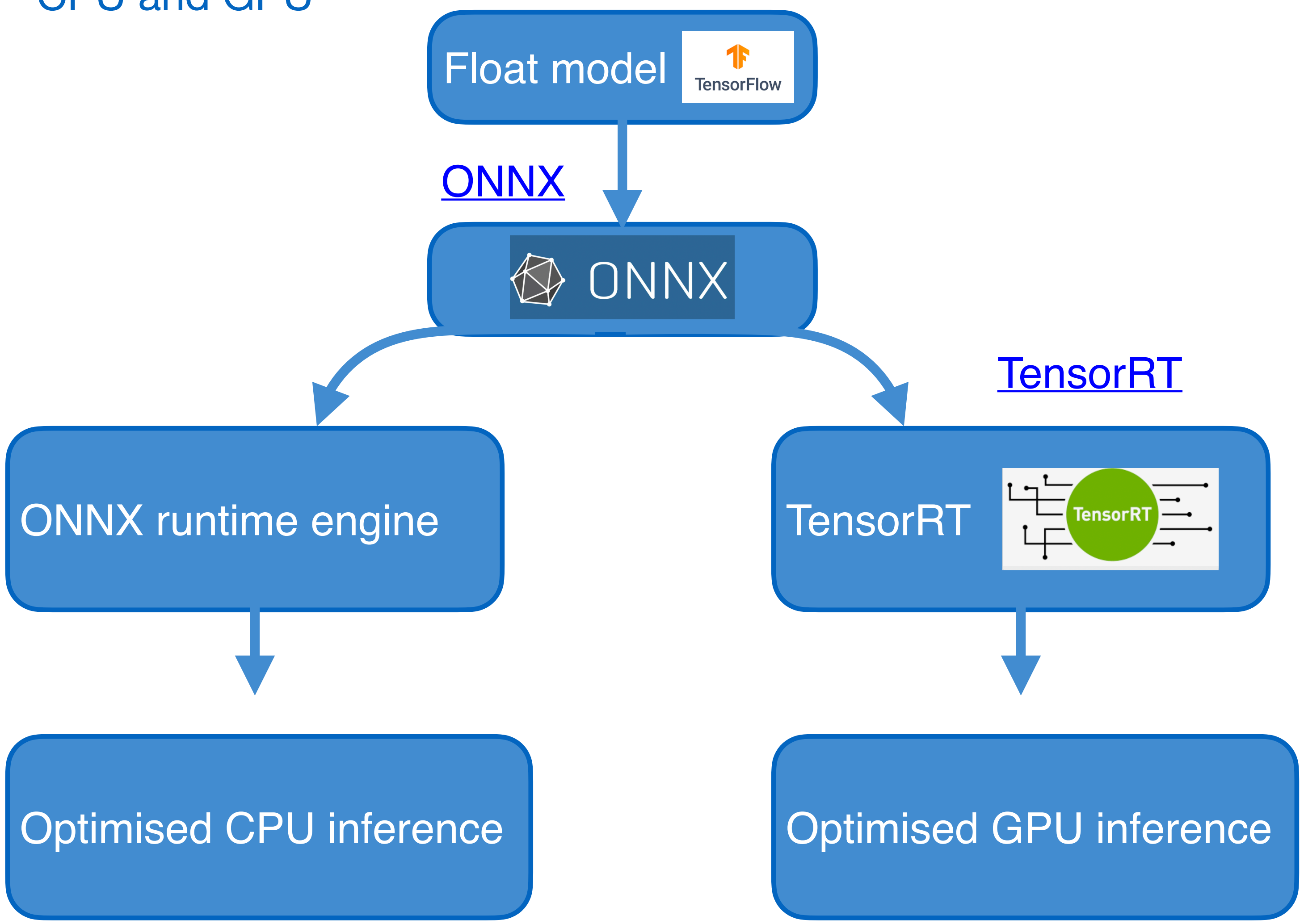
Based on AMD 7nm Versal
LUT: 900k
Off-Chip DDR memory: 16GB
Off-Chip DDR bandwidth: 102 GB/s
Network Interface: 2x QSFP28
C/C++ API also available
Cost is approximately: 10-15k euro

ML models: DNN, CNN and RNN

Note: support for GNN still missing

Application overview

CPU and GPU



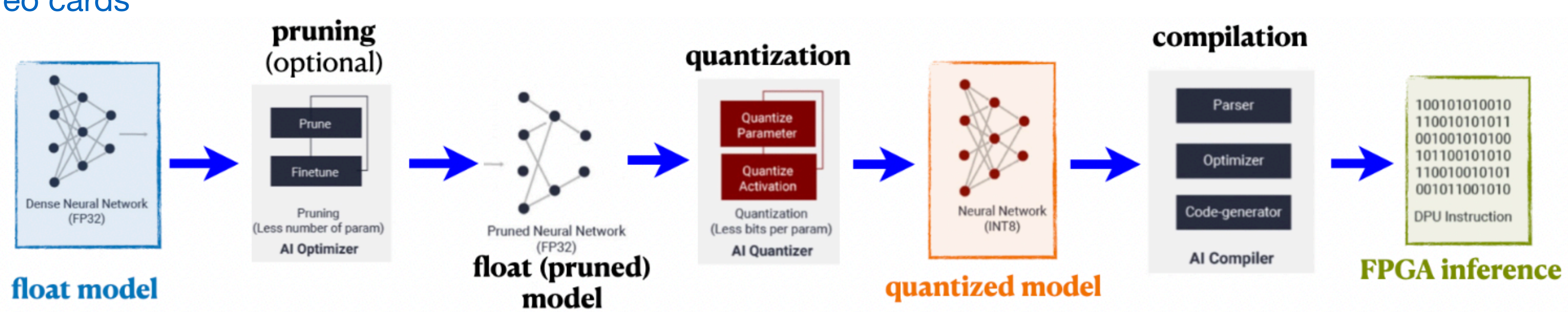
CPU and GPU Hardware used:

GPU: NVidia RTX A5000 board with 24GB of GDDR6 memory

CPU: single CPU server based on an AMD Epyc 7302 processor running at 2.9 GHz

Same CPU used as host for the accelerator cards

Alveo cards



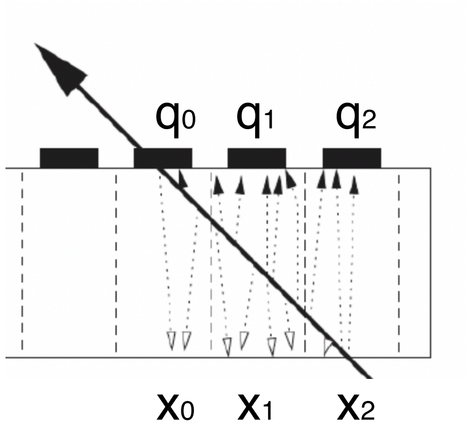
First application: cluster center position

A cluster is formed from neighbouring hits

Typically, the weighted centroid of the cluster is used

$$x_c = \frac{\sum_i x_i q_i}{\sum_i q_i}$$

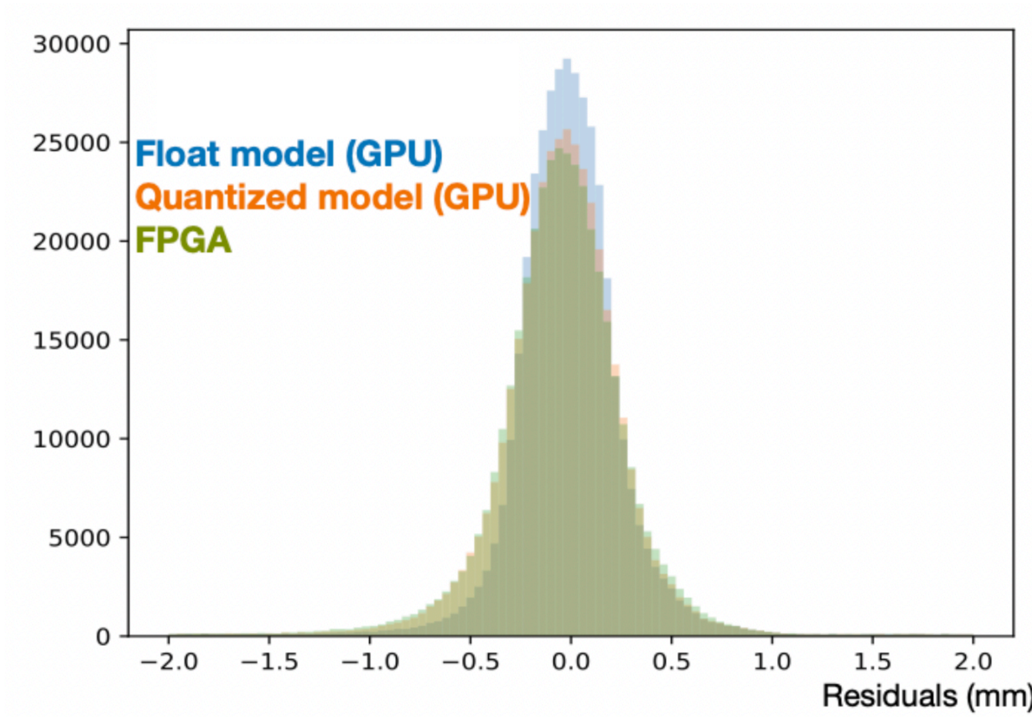
Collected charge: q_i
Strip position: x_i

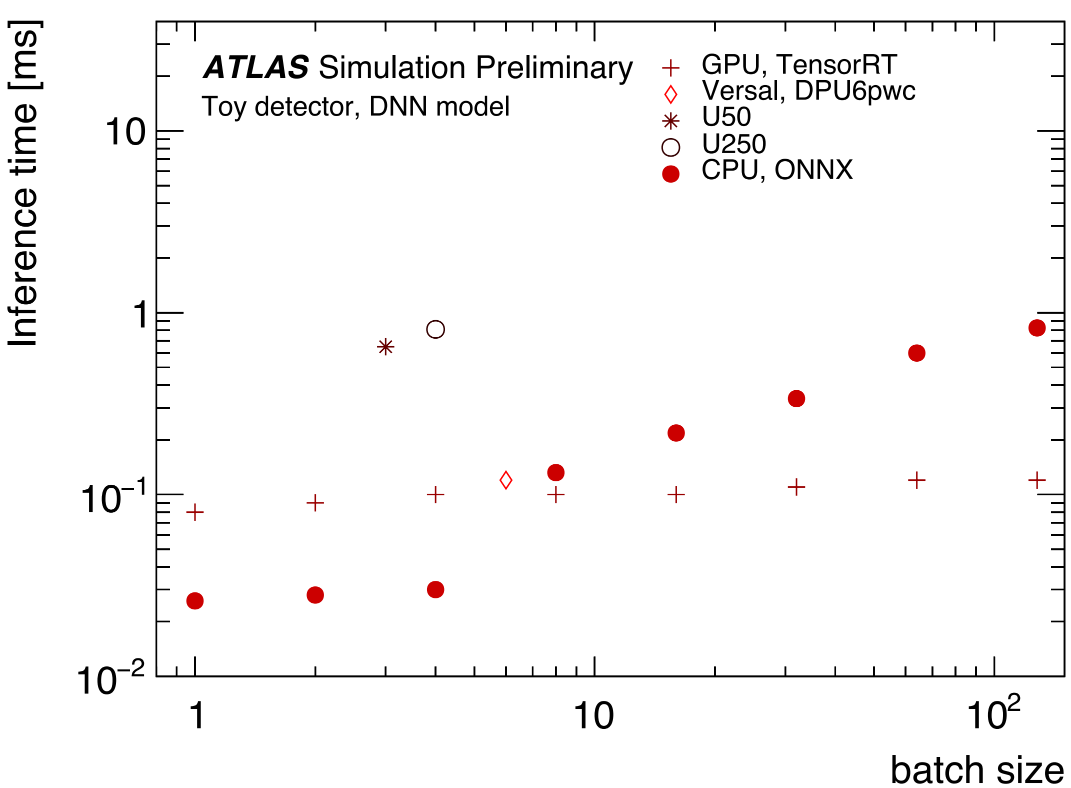


Depending on the incidence angle, collecting field, and magnetic field, the centroid estimation is inaccurate

A simple DNN based on ToT and spatial coordinate O(50k) parameters and 20 input variables. It improves up to 50% depending on the incident angle

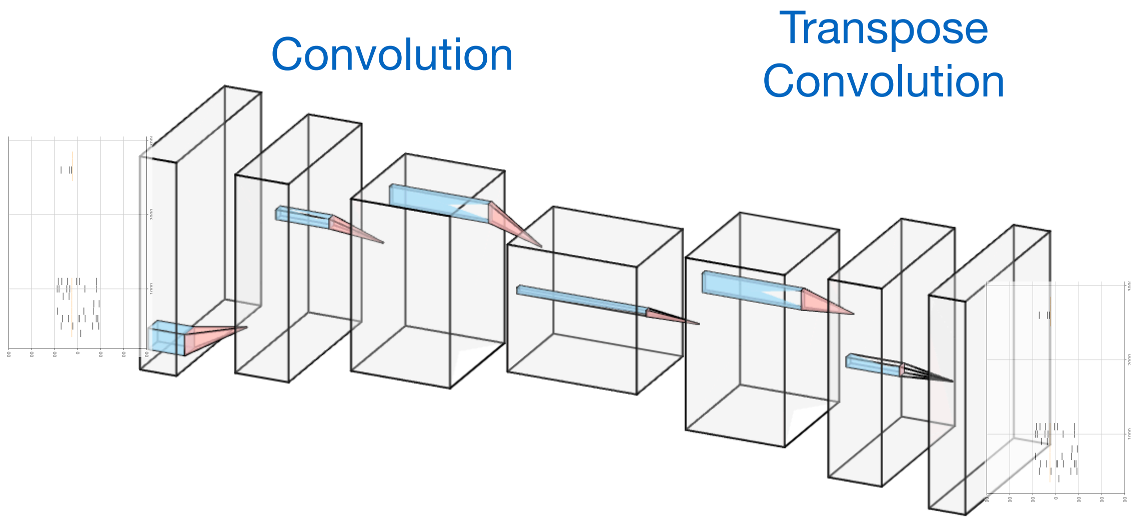
NB: inference time here are not a simulation, are real processing times obtained on the U50, U250 and Versal VCK5000

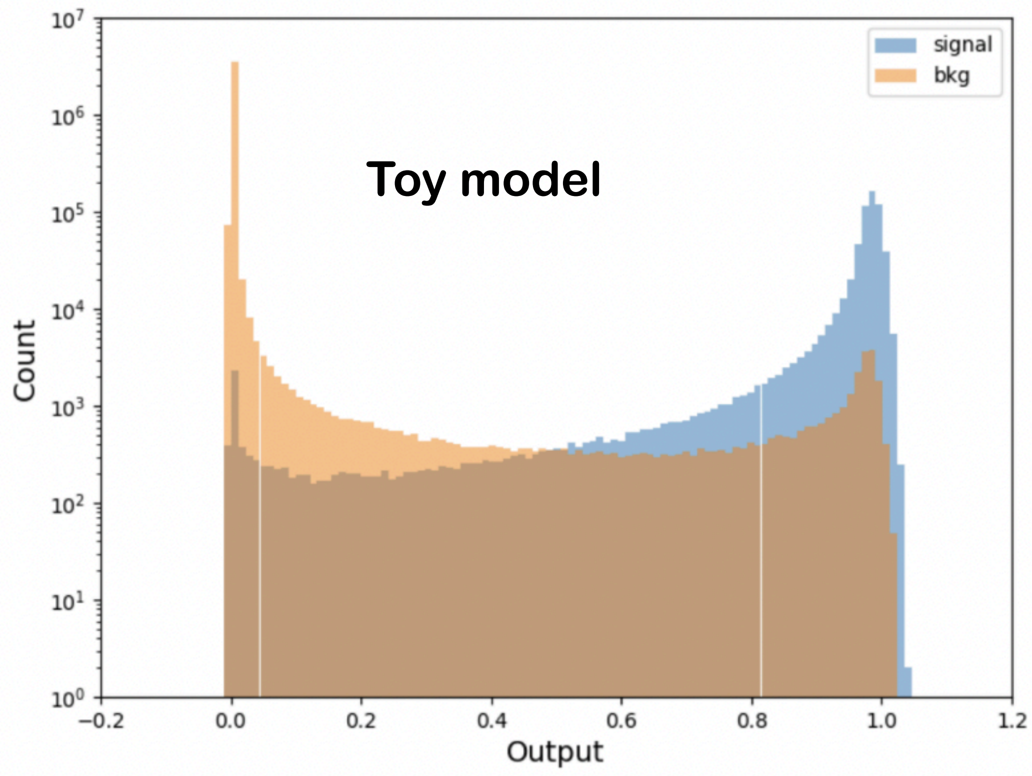


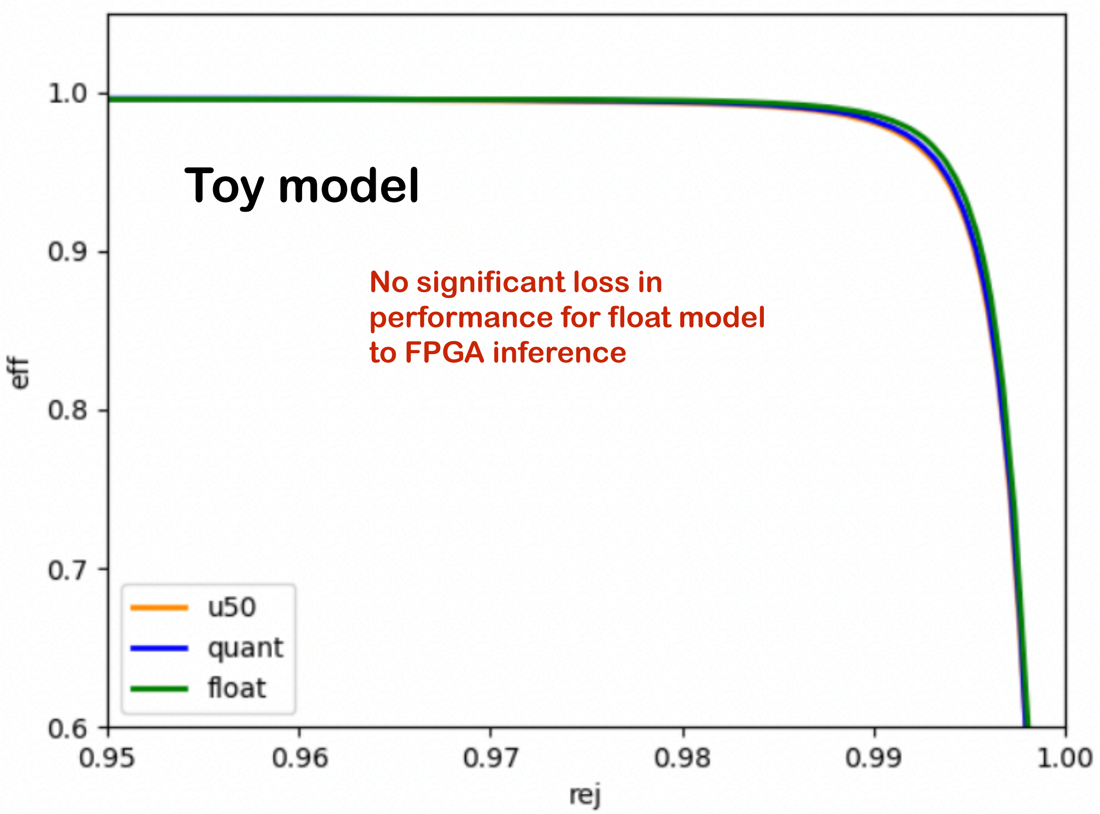


Alternative: a CNN approach

- In order to test the algorithm with Alveo cards, a CNN was also developed
- A CNN is not an optimal approach for pattern recognition tasks but it is useful for testing FPGA performance
- The number of parameters of the CNN model is O(50k)
- An event display is translated into a 3000x16 pixel 2D image, and convolution/deconvolution operation are used
- The output is an image whose intensity indicates the probability of the hits being associated to the muon

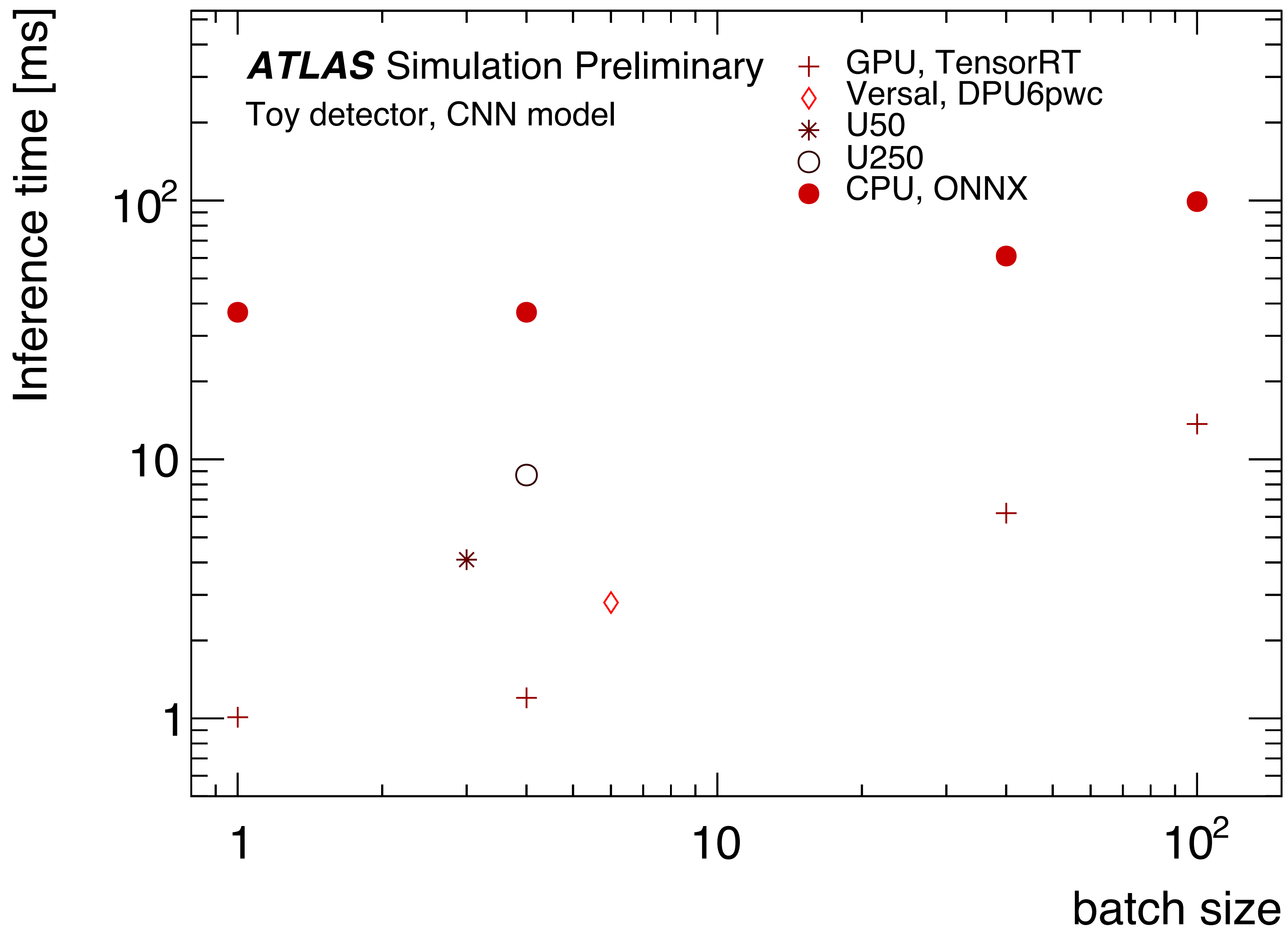






Comparison CNN

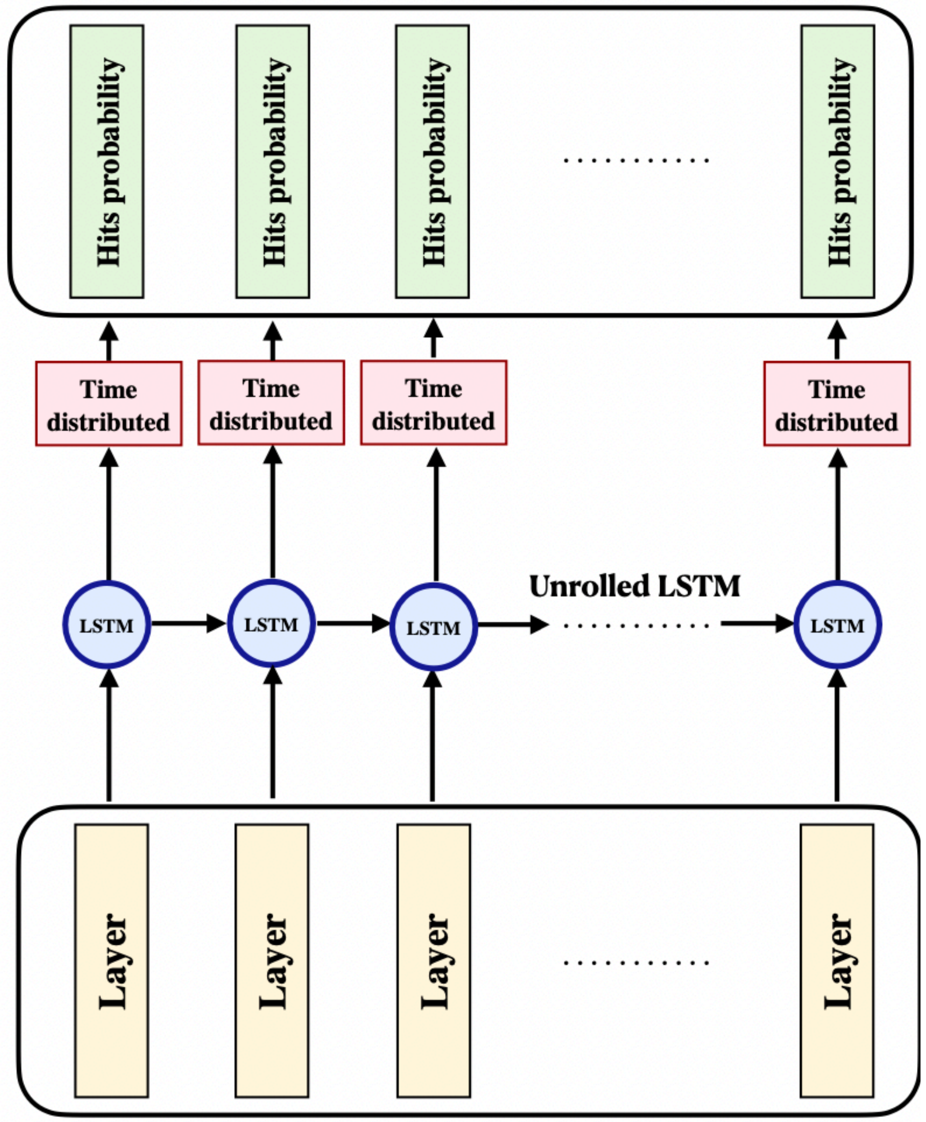
- CNN model successfully tested on CPU, GPU and several FPGAs
- Overall CPU already meets the requirement imposed by the HLT latency
- Largest improvement is seen with TensorRT on GPU.
- Study on CPU load will be performed, together with power dissipations
- Keep in mind this task does not need very deep CNN!



NB: no scaling for hit rate (single event occupancy) at inference time, as expected

Pattern recognition with an RNN

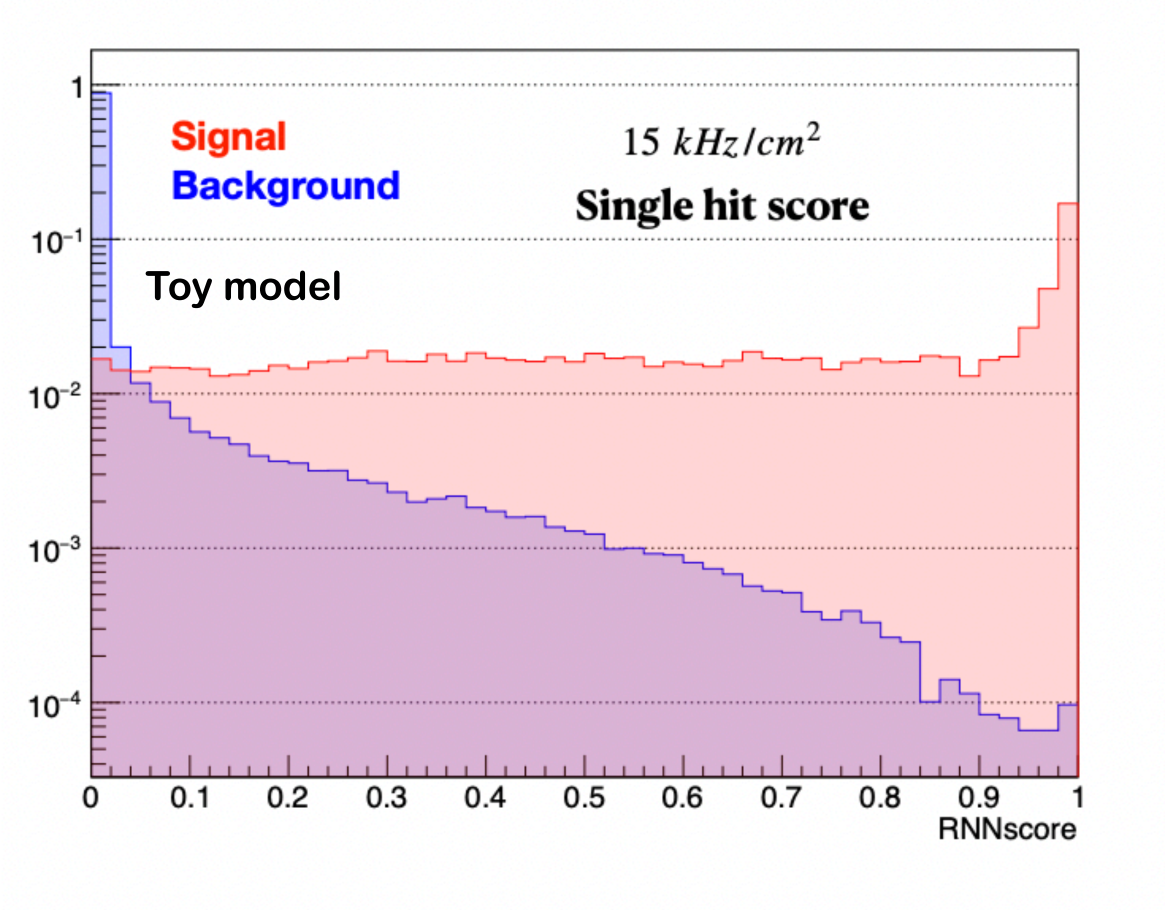
- Inputs are output of the previous DNN (position of the particle crossing within each cluster)
- Free parameter of the network O(300k)
- More sophisticated ML approaches such as GNN and/or transformers are not yet supported by Alveo cards
- In the RNN approach, consequent layers are ordered based on their position
- Three possibilities: outside-in or, inside-out, or also bidirectional
- Even if in principle supported, we failed to run RNN over the VCK500 card, tried Vitis-AI tag v3.5

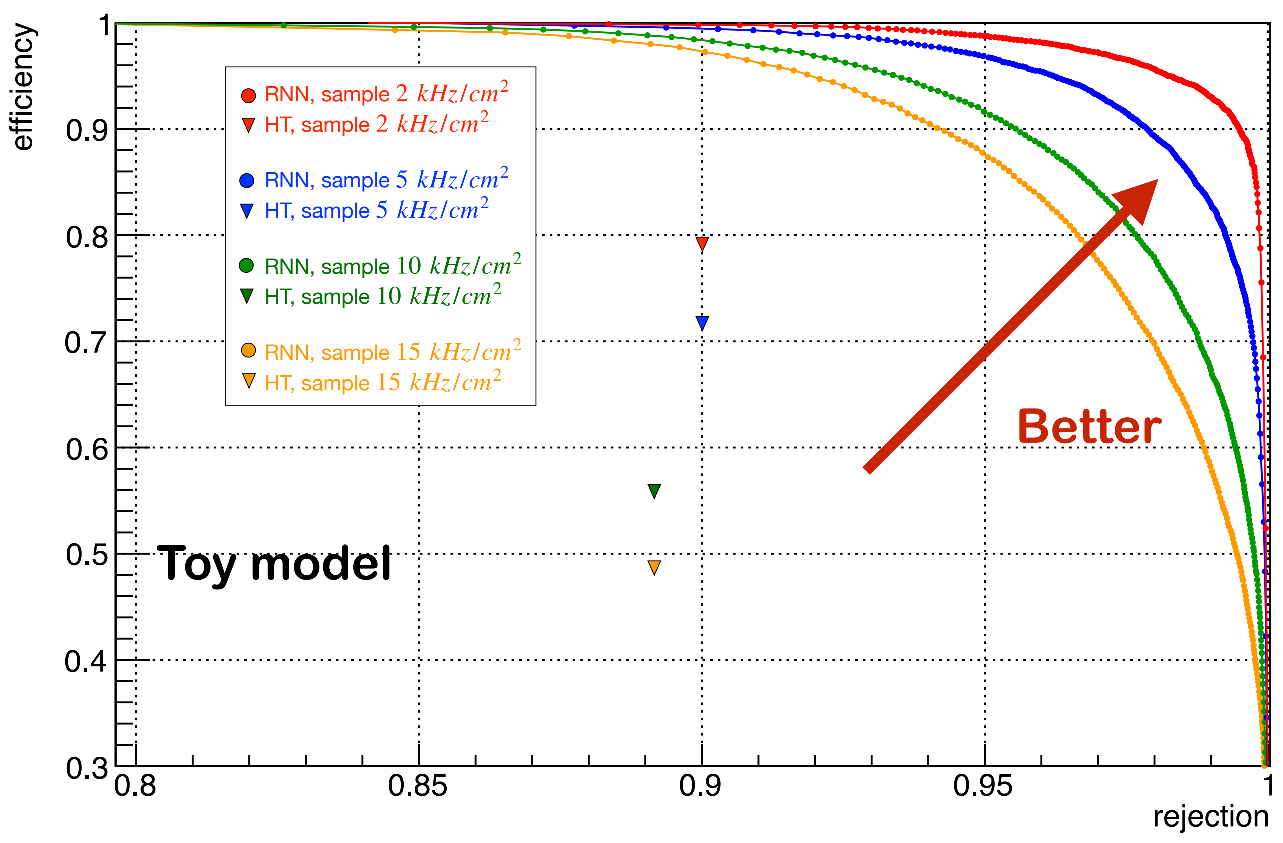


RNN performance results

Performance evaluated for different rates, generally, a decrease of performance is seen at higher rates, as expected

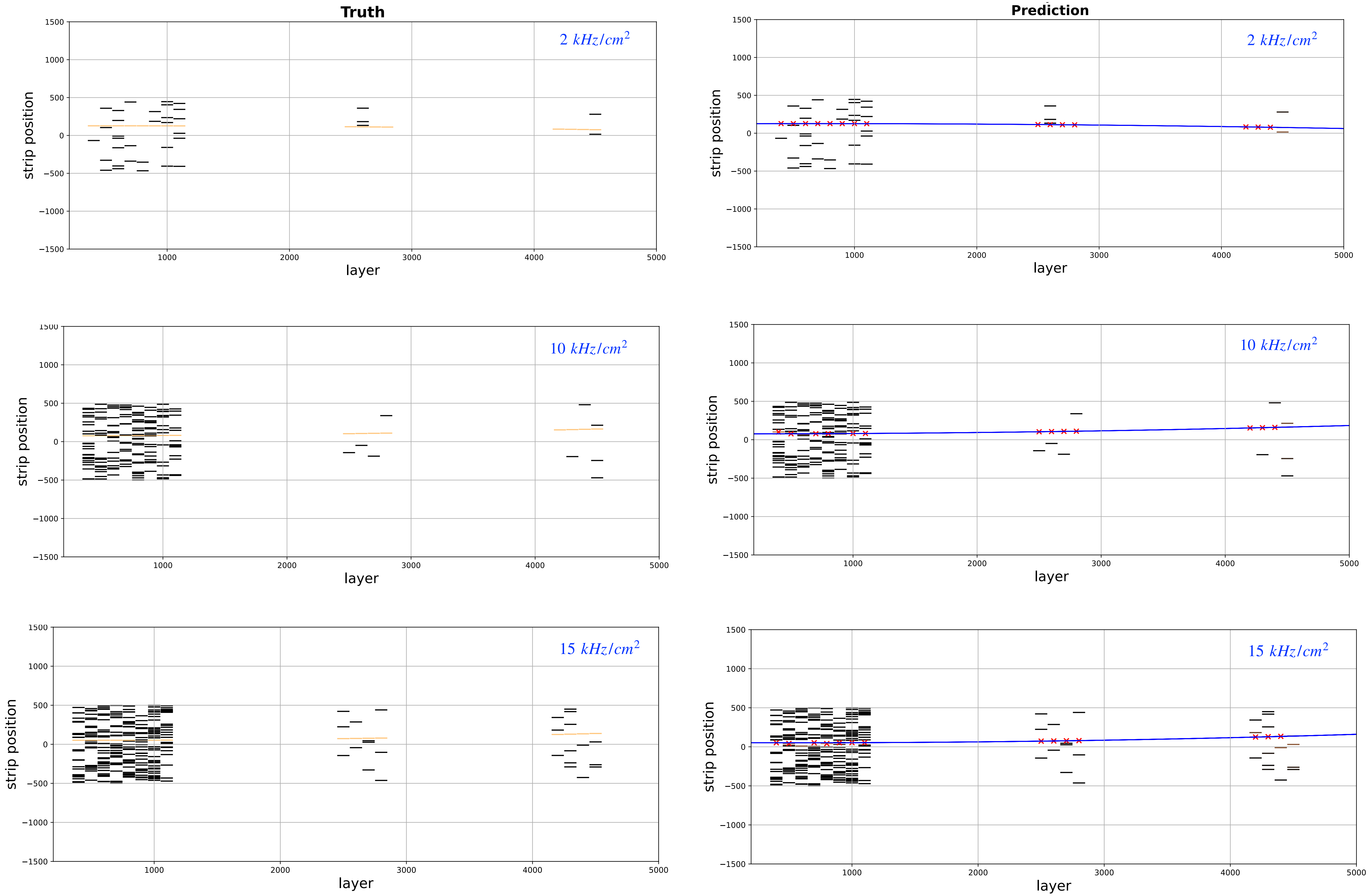
Hough Transform used as benchmark, NB: not very much fine tuned or optimised





Remember that this performance are based on a realistic toy, full implementation in ATLAS ongoing

Occupancy examples



RNN timing results

Tested on CPU (single core and multi-core) via ONNX, and on GPU with built-in tensor-flow and accelerated with TensorRT

NB: these numbers are from real ATLAS RUN3 data, not on the toy model.

Batch size = 1	Inference time (s)	ONNX CPU load/ core	GPU load
CPU 1 core	1.5E-03	100%	-
CPU 10 cores	1E-03	100%	-
GPU tensorRT	2E-02	-	23%

Batch size=1e3	Inference time (s)	ONNX CPU load/ core	GPU load
CPU 1 core	8.5E-01	100%	-
CPU 10 cores	1E-01	100%	-
GPU tensorRT	2E-02	-	50%

CPU load when running over GPU yet to be tested.
Test on VCK5000 work in progress.

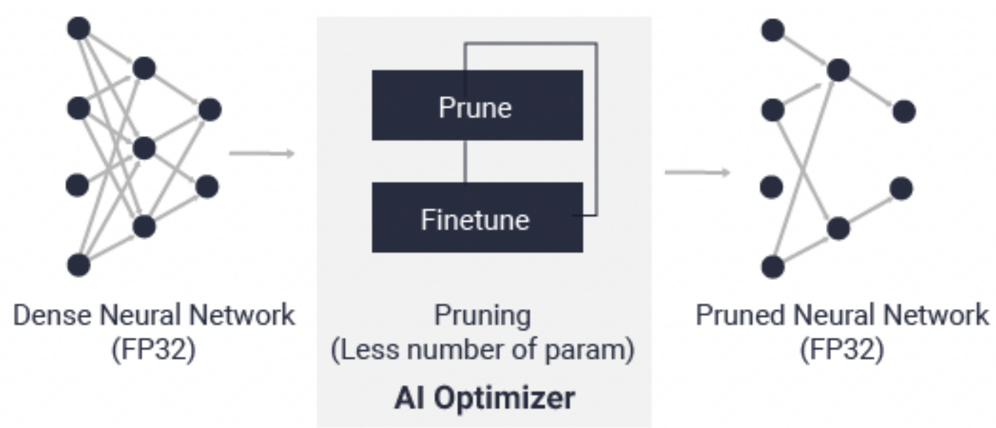
Conclusions

- Study on novel possibilities in the muon HLT algorithms
- Maintain good efficiency/rejections at high occupancy (good news for HL-LHC)
- DNN and CNN model successfully tested on three Alveo cards
- Inference time generally O(ms) and within latency requirement of HLT
- RNN implementation into FPGA is under way
- Once done, power consumption, and CPU load when running on each accelerators will be studied

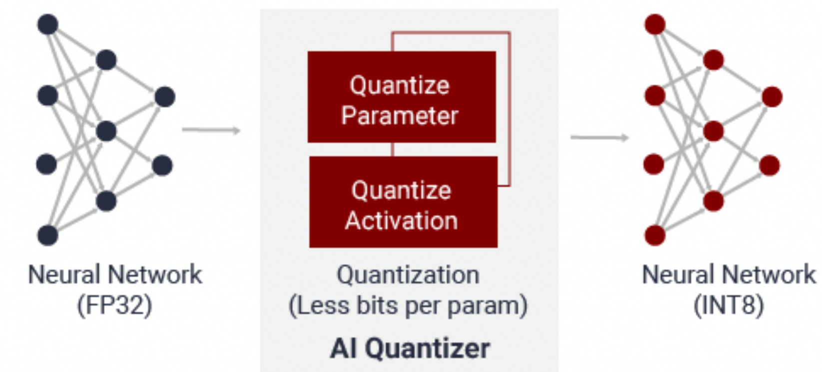
Workflow to use Alveo cards

Develop your favourite model with your favourite framework

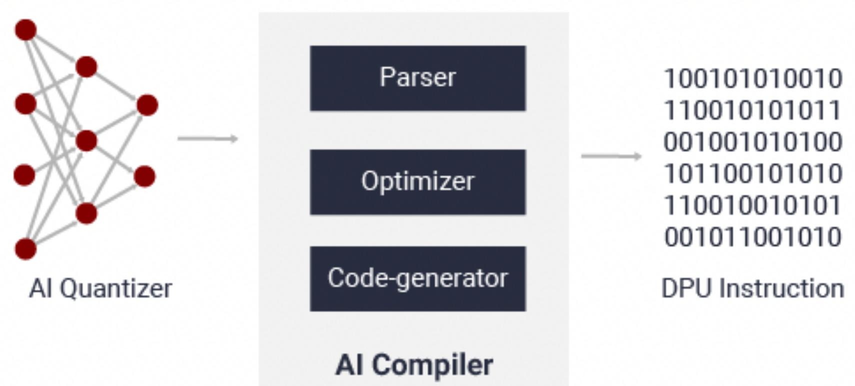
Pruning (optional)



Quantize



Compile



Run the inference

Varying the batch size

Main point is that tensorRT does not work with dynamic batch sizes

TensorFlow Model → ONNX model → New ONNX model with fix bs $\xrightarrow{\text{tensorRT}}$

```
import onnx
onnx_model = onnx.load_model('model_singleLoss.onnx')

BATCH_SIZE = 1
inputs = onnx_model.graph.input
for input in inputs:
    dim1 = input.type.tensor_type.shape.dim[0]
    dim1.dim_value = BATCH_SIZE

model_name = "model_singleLoss_mod.onnx"
onnx.save_model(onnx_model, model_name)
```

Models we are interested in

RNN:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 16, 100)]         0
_________________________________________________________________
lstm (LSTM)                  [(None, 16, 200), (None,  240800
_________________________________________________________________
lstm_1 (LSTM)                [(None, 16, 20), (None, 2 17680
_________________________________________________________________
time_distributed (TimeDistri (None, 16, 51)            1071
=================================================================
Total params: 259,551
Trainable params: 259,551
Non-trainable params: 0
```

CNN:

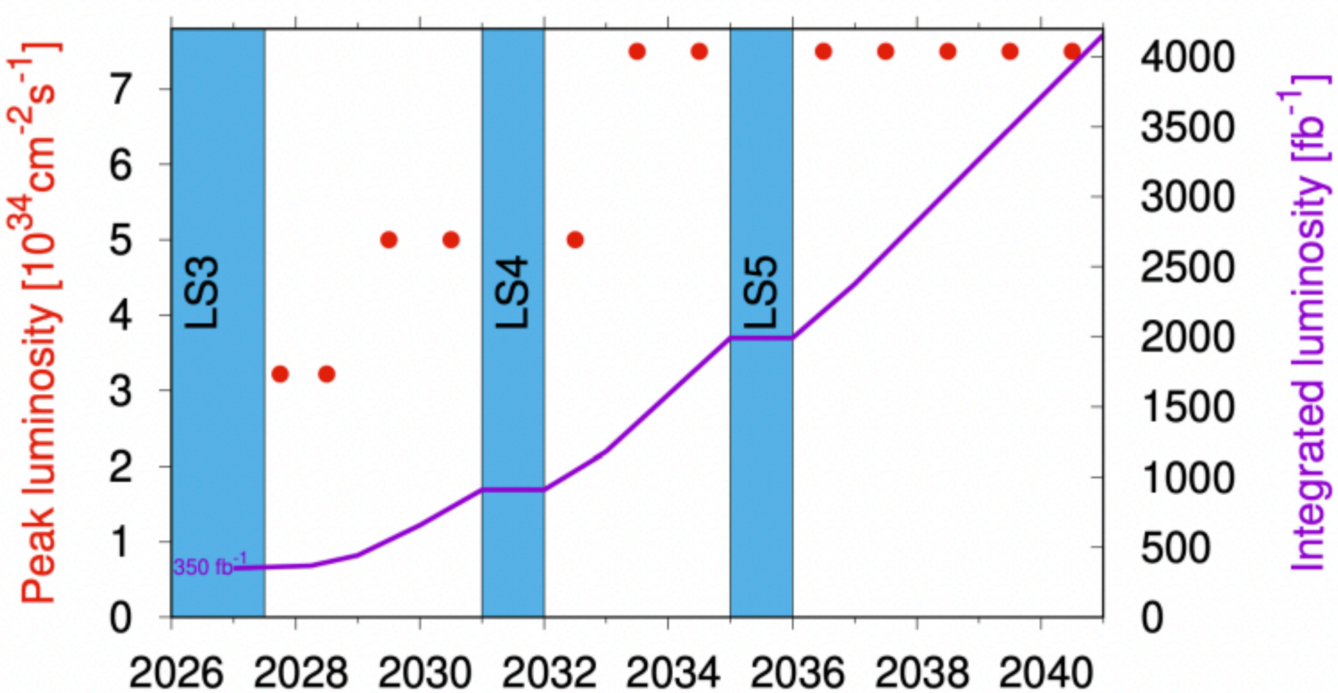
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 3000, 16, 1)]     0
_________________________________________________________________
conv1 (Conv2D)               (None, 3000, 16, 2)       194
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 1500, 16, 2)       0
_________________________________________________________________
conv2 (Conv2D)               (None, 1500, 16, 4)       100
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 750, 16, 4)        0
_________________________________________________________________
conv3 (Conv2D)               (None, 750, 16, 8)        392
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 375, 16, 8)        0
_________________________________________________________________
conv4 (Conv2D)               (None, 375, 16, 16)       6160
_________________________________________________________________
pool4 (MaxPooling2D)         (None, 375, 8, 16)        0
_________________________________________________________________
conv5 (Conv2D)               (None, 375, 8, 16)        32784
_________________________________________________________________
Tconv0 (Conv2DTranspose)     (None, 375, 16, 2)        258
_________________________________________________________________
Tconv1 (Conv2DTranspose)     (None, 750, 16, 4)        68
_________________________________________________________________
Tconv2 (Conv2DTranspose)     (None, 1500, 16, 8)       264
_________________________________________________________________
Tconv3 (Conv2DTranspose)     (None, 3000, 16, 16)      1040
_________________________________________________________________
output (Conv2D)              (None, 3000, 16, 1)       49
=================================================================
Total params: 41,309
Trainable params: 41,309
Non-trainable params: 0
```

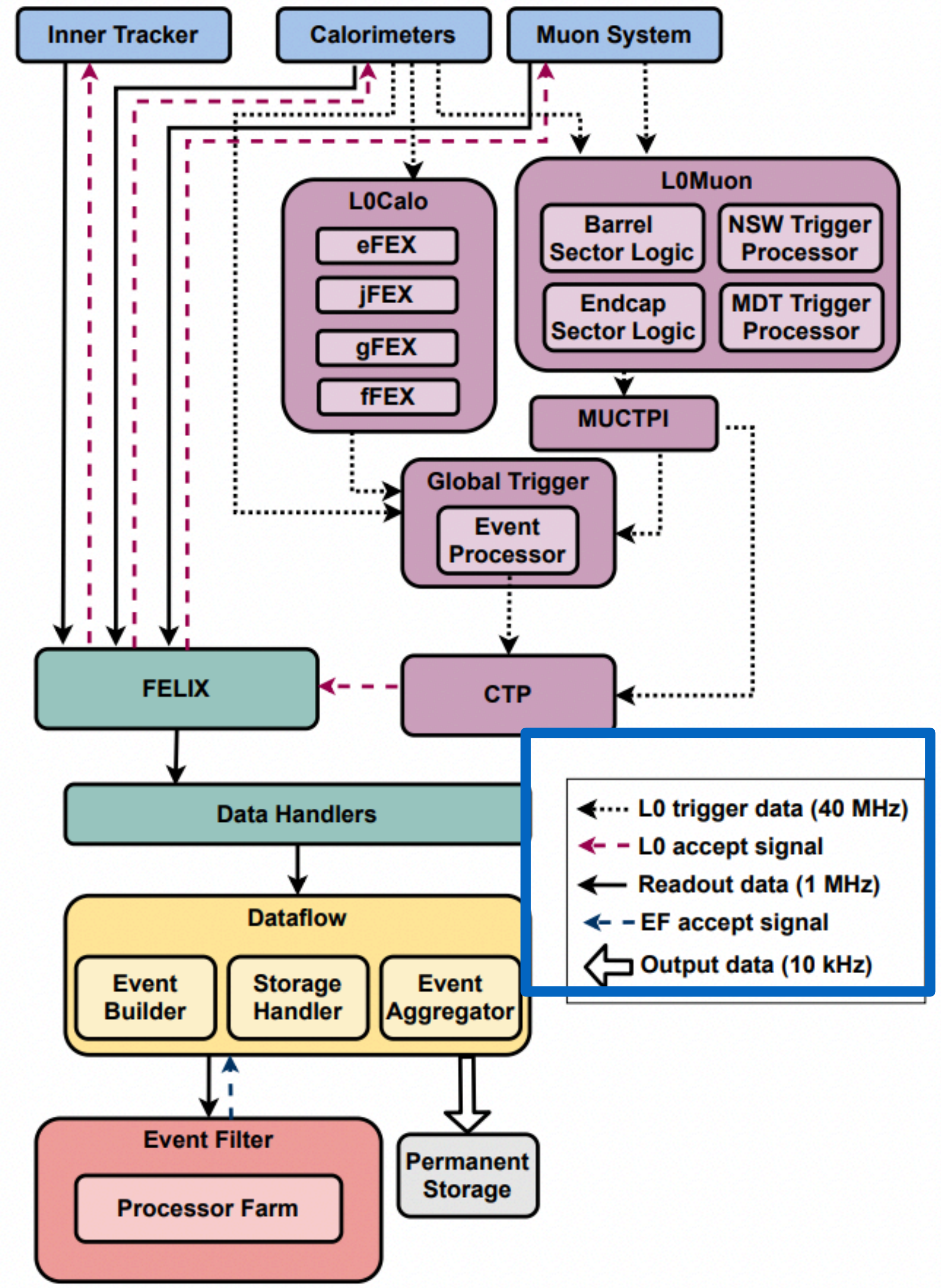
ATLAS HL-LHC trigger system

The work here is relevant for future RUN3 operations, but most importantly for triggering at HL-LHC

High luminosity and pile-up makes trigger decisions much more challenging

We will mostly consider as use-case muon system, for future applications to muon tracking





TDR trigger HL-LHC

The toy model used in this study

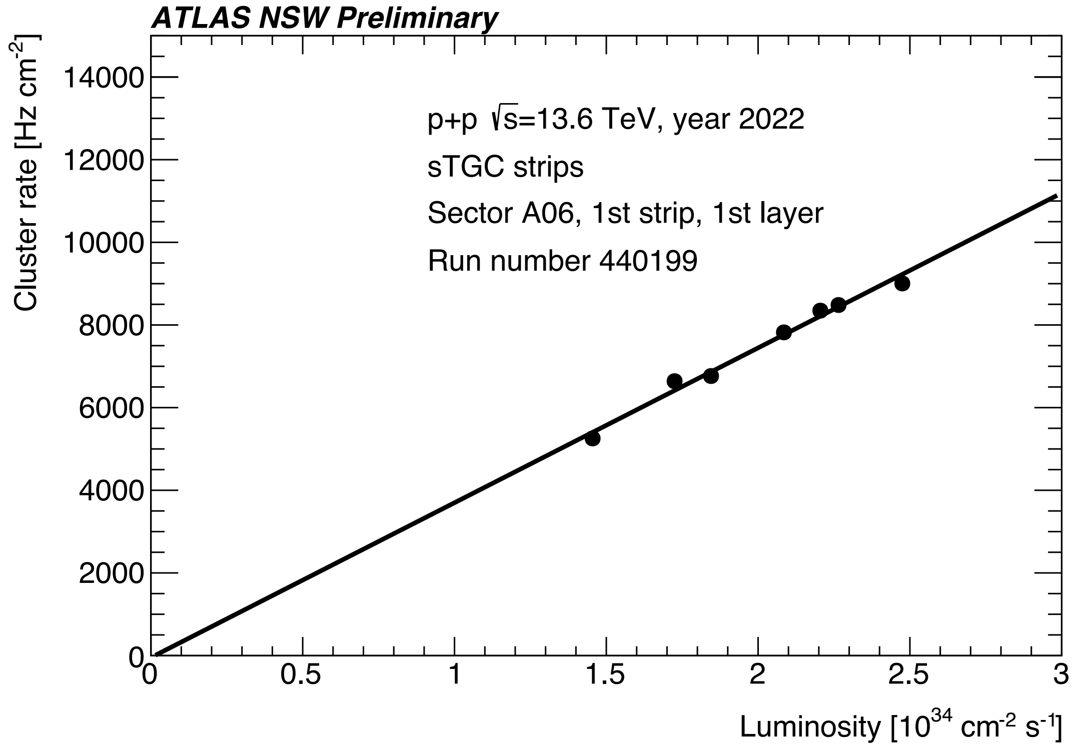
To speed up R&D part of the study, a toy model is simulated

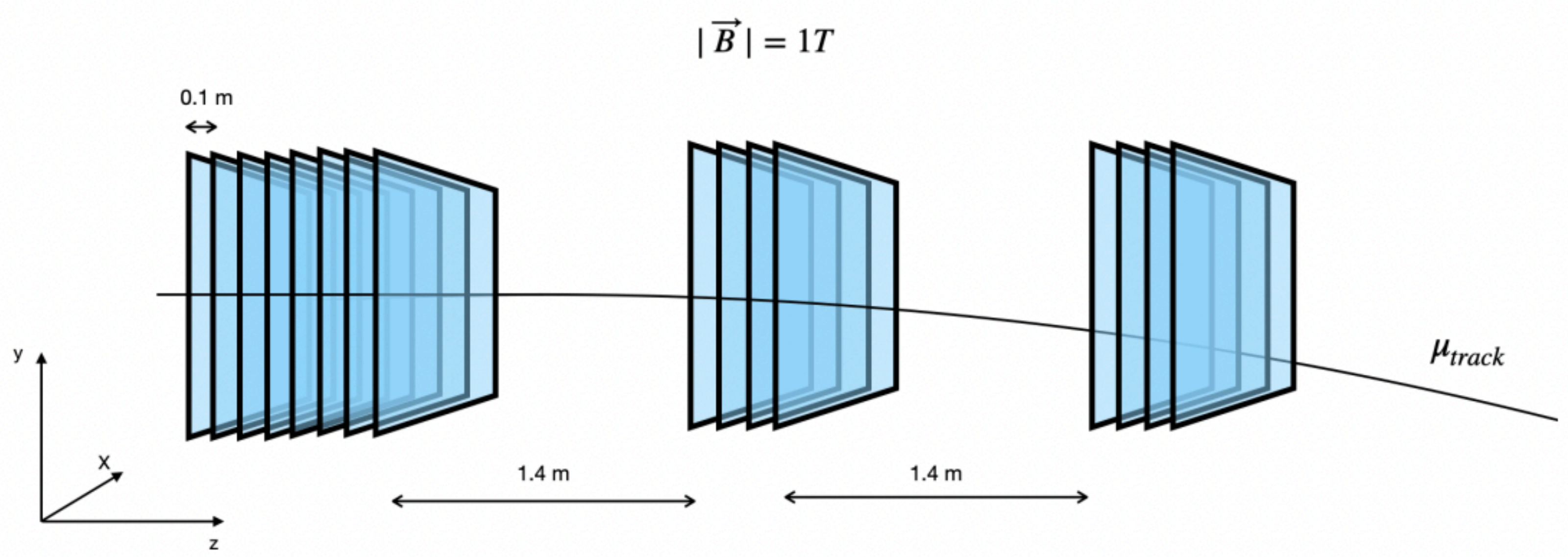
Toy model is inspired by a muon system

4 samples produced with different noise rates: 2, 5, 10, 15 kHz/cm**2

Effect from correlated background is also emulated

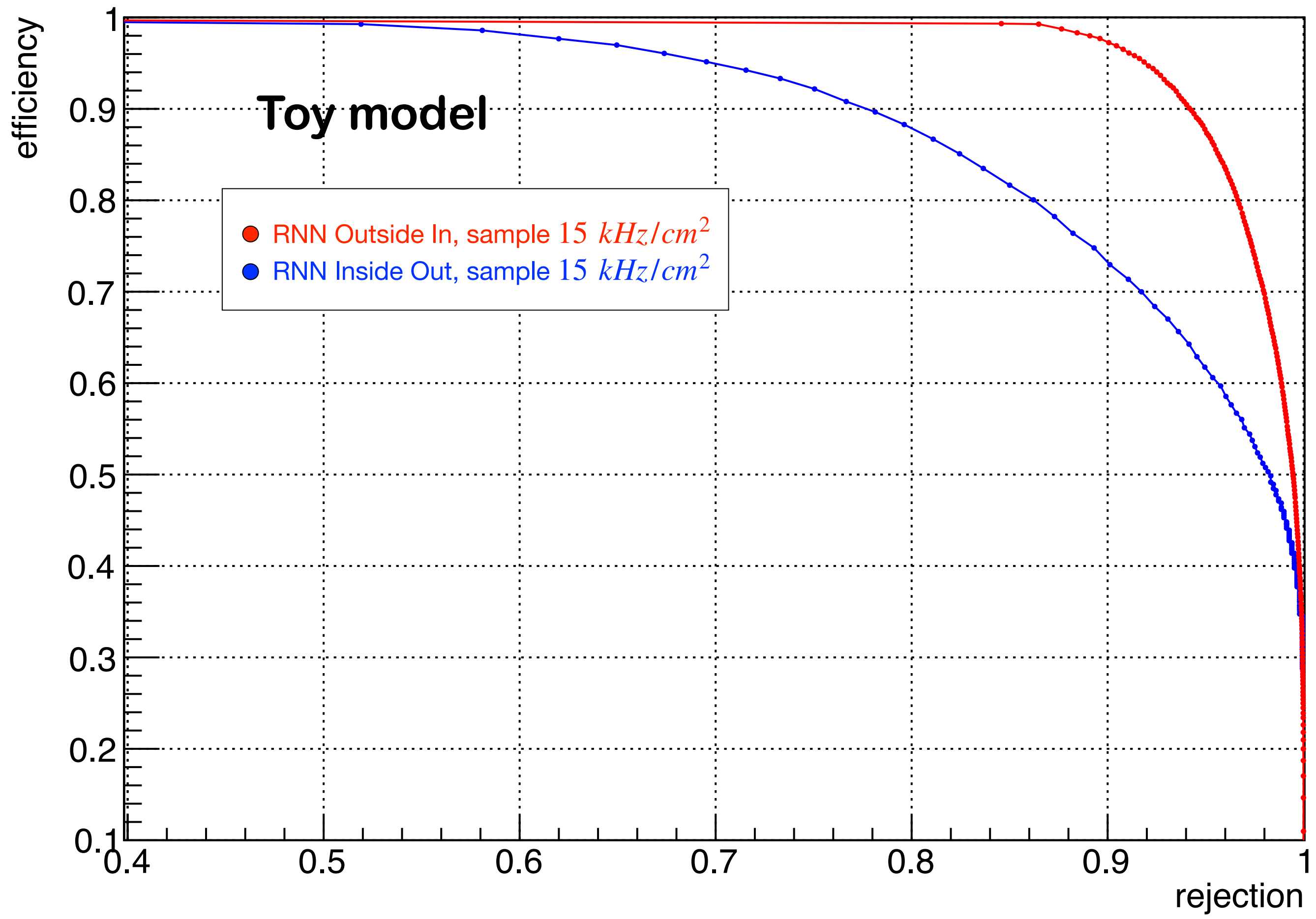
Will now discuss the main reco steps for tracking: clustering and pattern reco and their performance on CPU/GPU and FPGAs





RNN performance results

M. Carnesale
PhD Thesis



RNN models have inherently an order.

The initiation is to interpret them as

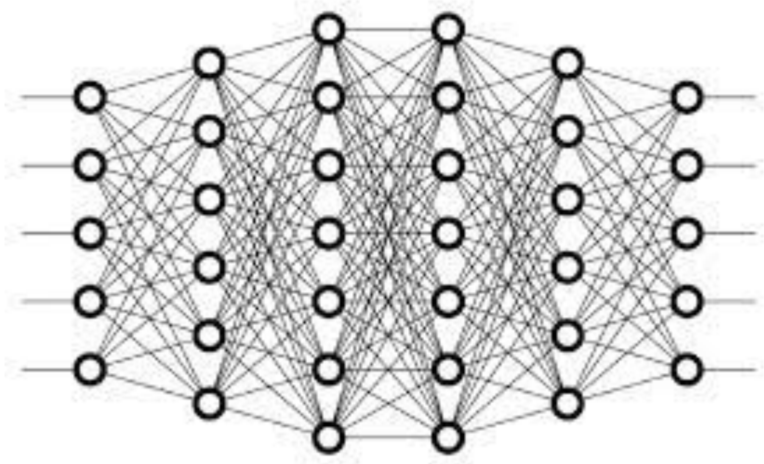
The Deep Neural Net approach

A deep neural network is used (similar to what done in the inner silicon tracker ref)

Inputs are:

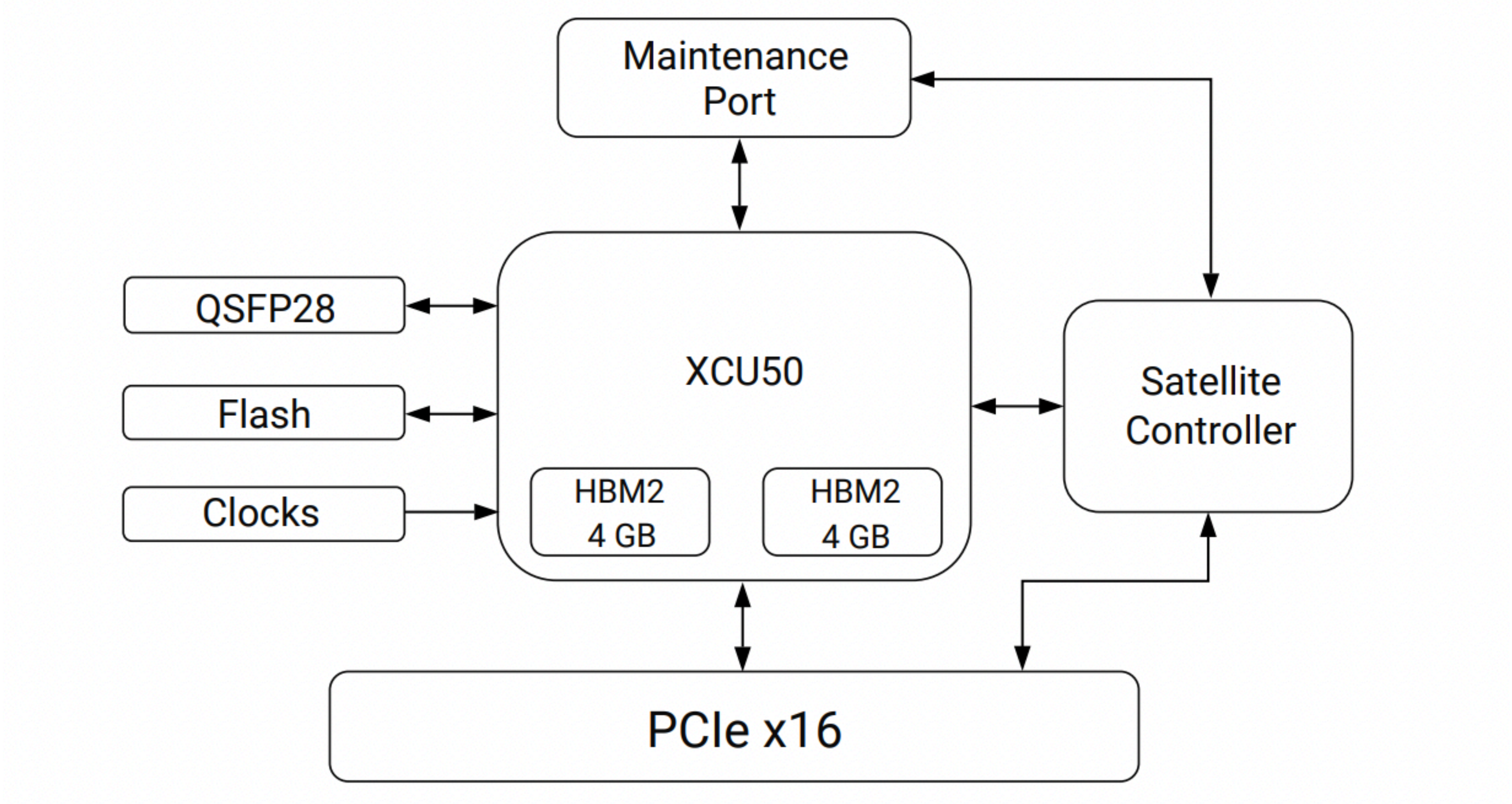
1. The total number of hits belonging to the cluster
2. The charge of the strip with highest charge
3. The charge of its two left-right closest neighbours
4. The position of the strip with highest charge
5. The Position of its two left-right closest neighbours

NB: if the cluster has less than 5 strips, zero-padding is employed



Standard regression using as target the true crossing position of the muon

U50



Maintenance Port
QSFP28
Flash
Clocks
XCU50
HBM2 4 GB
HBM2 4 GB
Satellite Controller
PCIe x16

U250

Figure: U200/U250 Block Diagram